(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 953 903 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
03.11.1999 Bulletin 1999/44

(51) Int. Cl.^{6}: G06F 9/46

(21) Application number: 99102487.8

(22) Date of filing: 10.02.1999

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 29.04.1998 GB 9809022

(71) Applicant:
International Computers Ltd.
London, EC2A 1DS (GB)

(72) Inventor:
Lysons, John Spencer Marshall
Audenshaw, Manchester M34 5EL (GB)

(74) Representative:
Guyatt, Derek Charles et al
International Computers Limited
Intellectual Property Department
Cavendish Road
Stevenage, Herts, SG1 2DY (GB)

(54) Semaphore for a computer system

(57) A computer system is described in which access to a shared resource is controlled by a semaphore. The semaphore comprises a lock value and a key value. When a process wishes to access the shared resource, it calls a reserve function which increments the lock value and stores the unchanged value of the lock value in a local variable. The reserve function then performs a loop in which it repeatedly compares the key value with the local variable until they equal. When the process has finished with the shared resource, it calls a release function which increments the key value. This guarantees to allocate the semaphore to processes in the order in which the processes requested it (i.e. in chronological order), in a cheap and effective manner.

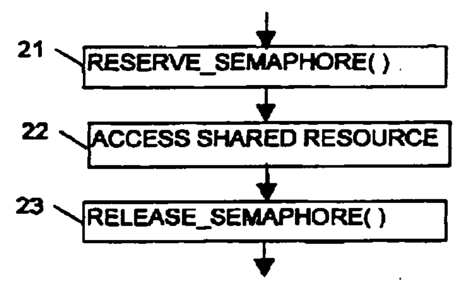

FIG. 2

EP 0 953 903 A2

Description

Background to the Invention

[0001] This invention relates to semaphores for computer systems.

[0002] A semaphore is a mechanism for controlling access to a shared resource by a number of concurrent processes. The shared resource may, for example, be an area of shared memory. The concurrent processes may run on a single processor, or on different processors.

[0003] In one known form of semaphore, referred to as a spinlock semaphore, each shared resource has a semaphore bit associated with it. Whenever a process wishes to access a shared resource, it tests the semaphore bit associated with the resource. If the semaphore bit is set, the process "spins" in a tight loop, in which it repeatedly tests the semaphore value. When the process finds that the semaphore bit is reset, it sets the semaphore bit and proceeds to access the shared resource. The testing and setting of the semaphore bit must be performed as a single atomic (i.e. indivisible) operation. When the process has finished accessing the shared resource, it resets the semaphore bit.

[0004] A problem with this conventional spinlock semaphore, however, is that it cannot guarantee to allocate the semaphore to the processes in the order in which the processes requested it. In particular, it is possible for two processes to continually swap a resource between them, thereby blocking a third process. The object of the present invention is to provide an improved spinlock semaphore that is guaranteed to allocate the semaphore in the order in which the processes requested it.

Summary of the Invention

[0005] According to the invention, a method of controlling access by a plurality of processes to a plurality of shared resources in a computer system comprises:

(a) providing each of the shared resources with a first semaphore value and a second semaphore value;

(b) when a process requires to access a shared resource, storing a reservation number as a local variable for the process, the reservation number being derived from the first semaphore value of the resource, updating the first semaphore value of the resource in a predetermined manner, performing a loop which repeatedly compares the reservation number stored for the process with the second semaphore value of the resource, and permitting the process to access the resource when the reservation number stored for the process is in a predetermined relationship with the second semaphore value of the resource; and

(c) when a process has finished accessing a shared resource, updating the second semaphore value of the resource in a predetermined manner.

Brief Description of the Drawings

[0006]

Figure 1 is a schematic diagram showing a computer system embodying the invention.

Figure 2 is a flow chart showing part of a client process.

Figure 3 is a flow chart of a Reserve_semaphore function.

Figure 4 is a flow chart of a Release_semaphore function.

Description of an Embodiment of the Invention

[0007] One semaphore mechanism in accordance with the invention will now be described by way of example with reference to the accompanying drawings.

[0008] Figure 1 shows a computer system 10, which runs a number of client processes 11. The system may comprise a single processing unit with a multi-threading operating system allowing a number of processes to run concurrently. Alternatively, the system may be a multiprocessor system, in which different processes can run simultaneously on different processors.

[0009] The client processes 11 share a number of resources 12, such as memory. Each resource has a semaphore 13 associated with it. The semaphore comprises a lock value 14 and a key value 15. The lock and key values are shared variables, and in this embodiment are initially set to the same value.

[0010] Two functions are provided for managing the semaphores: a Reserve_semaphore() function 16 and a Release_semaphore() function 17. These functions can be called by the client processes.

[0011] Figure 2 shows part of one of the client processes which makes use of the semaphore.

[0012] (Step 21) When the client process wishes to access a shared resource, it calls the Reserve_semaphore() function. The call contains a parameter, specifying the identity of the semaphore associated with the resource in question.

[0013] (Step 22) When this call returns, the client process accesses the shared resource.

[0014] (Step 23) The client process then calls the Release_semaphore() function.

[0015] Figure 3 shows the Reserve_semaphore() function.

[0016] (Step 31) The function increments the lock value of the semaphore, and sets a local variable orig_lock equal to the unincremented value of the lock. This must be done as a single indivisible action. For